**ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

**--------------------------------------------------------------------------**

A red and yellow sign with a yellow and white background

AI-generated content may be incorrect.

**Game Pokemon đối kháng**

**Học phần: Hệ Nhúng**

**GVHD:** TS. Đỗ Công Thuần

**Nhóm 4**

1. **Nguyễn Khắc Tiếp 20225767**
2. **Nguyễn Văn Hoàn 20225708**
3. **Nguyễn Việt Thành 20225931**

**Hà Nội, 11 tháng 6 năm 2025**

MỤC LỤC

[**1. Giới thiệu đề tài**](#_ue97moy5a4gq) **2**

[- Mô tả về sản phẩm/mục tiêu của project:](#_ukotqmvxzkf6) 2

[- Cốt truyện của trò chơi:](#_lkqnk54goras) 2

[- Các yêu cầu đối với project:](#_c5iz6tsfro0x) 3

[**2. Thiết kế**](#_tudy6uddihc6) **4**

[- Phân chia chức năng:](#_9oey4f28m3df) 4

[+) Phần cứng:](#_19e6k9tqr9ll) 4

[+) Phần mềm:](#_h3tv1rtz21eh) 4

[- Thiết kế phần cứng:](#_x5m48ghobqgr) 5

[+) Linh kiện: nút bấm PA0, joystick](#_upfcameh0am0) 5

[+) Sơ đồ mạch:](#_ntcoofxwdhui) 5

[+) Sơ đồ ghép nối:](#_xsp54rczmpir) 6

[+) Tốc độ trao đổi dữ liệu:](#_73cfqxklm4mg) 6

[- Thiết kế phần mềm:](#_bof2ku7xz1g4) 6

[+) Module điều khiển joystick:](#_bejizi36ph60) 6

[+) Module xử lý logic game:](#_qzz3i2vz5d67) 11

[+) Module xử lý các cấp độ level:](#_q54stb4c3umo) 30

[**3. Cài đặt/Xây dựng hệ thống**](#_7vkpxu7y1rs3) **40**

[- Mô tả các module phần mềm chính:](#_bu1yn7abfjna) 40

[- Đóng góp của từng thành viên](#_1t7qg52a2hl2) 40

[- Đánh giá:](#_d30xb14a82bi) 40

# 1. Giới thiệu đề tài

## - Mô tả về sản phẩm/mục tiêu của project:

Đề tài hướng tới việc xây dựng một trò chơi Pokémon đối kháng đánh theo lượt, cung cấp trải nghiệm người chơi qua ba cấp độ khó khác nhau: dễ (easy), trung bình (medium), khó (hard). Trò chơi sử dụng board phát triển STM32F429DISCO1 kết hợp với các nút bấm vật lý (switch), joystick để tương tác trực tiếp, điều khiển nhân vật trong game.

## - Cốt truyện của trò chơi:

**Pokémon: Cuộc Chiến Định Mệnh**

Tại ngôi làng nhỏ Serenity, nổi tiếng với các nhà huấn luyện Pokémon tài năng, hai huấn luyện viên trẻ, **Ashen** (áo đỏ) và **Sylvan** (áo xanh lá), đã lớn lên cùng nhau như hai người bạn thân thiết. Tuy nhiên, trong lòng mỗi người đều nung nấu giấc mơ lớn: trở thành nhà vô địch vĩ đại nhất trong lịch sử Serenity.

Ashen là người huấn luyện đầy nhiệt huyết, luôn coi Pokémon như những người bạn thân thiết, tin tưởng vào sức mạnh tình cảm và sự dũng cảm. Trong khi đó, Sylvan là chiến lược gia điềm tĩnh, tin vào khả năng phân tích, chuẩn bị kỹ lưỡng từng trận chiến, và sử dụng chiến thuật sắc bén để khai thác điểm yếu của đối thủ.

Cuối cùng, ngày mà cả làng Serenity mong đợi cũng đến: **Giải đấu Pokémon thường niên Serenity**. Trận chung kết trở thành cuộc đối đầu được mong chờ nhất giữa hai người bạn thời thơ ấu, Ashen và Sylvan.

Ashen sử dụng Pokémon của mình, **MegaRayquaza**—một Pokémon dũng cảm và đầy lửa nhiệt huyết. Còn Sylvan chọn **MegaX**—một Pokémon thông minh với khả năng kiểm soát thiên nhiên điêu luyện. Trận chiến diễn ra vô cùng căng thẳng, từng lượt đánh, từng đòn kỹ năng đều được tính toán kỹ càng.

MegaRayquaza, dưới sự chỉ huy của Ashen, chiến đấu bằng tất cả nhiệt huyết và lòng trung thành. Những đòn tấn công mạnh mẽ, quyết liệt của MegaRayquaza khiến khán giả liên tục hò reo.

Ngược lại, MegaX, với khả năng kiểm soát môi trường và sức mạnh chiến thuật, liên tục né tránh và tung ra các đòn phản công bất ngờ, làm suy yếu dần sức mạnh của đối thủ.

Sau một trận chiến kéo dài, hai Pokémon đều thấm mệt, hơi thở hổn hển nhưng ánh mắt vẫn kiên cường. Trong khoảnh khắc quyết định cuối cùng, cả Ashen và Sylvan đồng thời ra lệnh cho Pokémon tung đòn quyết định mạnh nhất.

Cả sân đấu như ngừng lại trong giây phút hai luồng sức mạnh chạm nhau dữ dội. Nhưng chỉ một khoảnh khắc sau, MegaRayquaza gục xuống, kiệt sức sau khi nỗ lực hết mình.

MegaX đứng vững, tuy kiệt quệ nhưng vẫn còn sức chiến đấu. Sylvan chính thức giành chiến thắng.

Ashen cúi đầu, nhẹ nhàng đến bên Pokémon của mình, mỉm cười tự hào vì MegaRayquaza đã chiến đấu hết mình. Sylvan bước tới, bắt tay bạn mình, tuyên bố rằng chiến thắng này không chỉ thuộc về cá nhân anh, mà là chiến thắng của tình bạn, lòng dũng cảm và quyết tâm của cả hai người.

Dân làng vỡ òa, ăn mừng chiến thắng của Sylvan, nhưng đồng thời cũng ghi nhớ sự dũng cảm và lòng quả cảm mà Ashen cùng MegaRayquaza đã thể hiện.

Từ đó, trận chiến này trở thành huyền thoại của Serenity, nhắc nhở mọi người rằng dù chiến thắng chỉ có một, nhưng tinh thần chiến đấu và tình bạn là điều tồn tại mãi mãi.

## - Các yêu cầu đối với project:

+) Yêu cầu chức năng:

* Giao diện 3 màn hình rõ ràng: Màn hình StartGame để bắt đầu trò chơi, Map di chuyển 2 nhân vật để bắt đầu trận đấu, Fight hiển thị thông tin 3 mức độ chơi (easy, medium, hard) và thông tin của 2 Pokemon gồm tên, thanh máu (HP), con trỏ thay đổi lượt, chiêu thức, items.
* Chế độ chơi turn-based rõ ràng, lần lượt theo thứ tự người chơi và máy.
* Ba mức độ chơi rõ ràng về độ khó, thay đổi theo chiêu thức và tỉ lệ dính các hiệu ứng.
* Sử dụng nút bấm vật lý/joystick để di chuyển nhân vật ở màn hình MAP và thao tác như con chuột để lựa chọn các hành động (tấn công, phòng thủ, kỹ năng đặc biệt).
* Hiệu ứng chiêu thức phù hợp từng chiêu thức trong game.
* Đầu ra hiển thị trên màn hình LCD trên board STM32F429DISCO1.

+) Yêu cầu phi chức năng:

* Trò chơi hoạt động ổn định, không bị lỗi hay giật lag khi sử dụng nút bấm/joystick.
* Dễ dàng phát triển và sửa, bảo trì code.

# 2. Thiết kế

## - Phân chia chức năng:

### +) Phần cứng:

* STM32F429DISCO1 làm mạch xử lý chính.
* Joystick di chuyển nhân vật ở màn Map, điều khiển con chuột ở các màn còn lại

+ nút bấm PA0 để click chọn.

* Màn hình LCD hiển thị đồ hoạ và trạng thái game.

### +) Phần mềm:

* Xử lý logic game theo từng mức độ khó.
* Xử lý hiệu ứng hình ảnh Pokemon, hiệu ứng tung chiêu.
* Xử lý module Groove Joystick

## - Thiết kế phần cứng:

### +) Linh kiện: nút bấm PA0, joystick

### +) Sơ đồ mạch:



### +) Sơ đồ ghép nối:

Joystick:

* Chân X -> PC3
* Chân Y -> PA5
* VCC -> 3V
* GND

### +) Tốc độ trao đổi dữ liệu:

* Joystick

- ADC được cấu hình với:

• Prescaler: PCLK2 / 4 = 90 MHz / 4 = 22.5 MHz

• Sampling Time: 3 chu kỳ

• Tổng thời gian chuyển đổi: T\_conv = (3 + 12) / 22.5 MHz ≈ 0.667 µs

- Ở các màn hình Start và Map: đọc ADC trong Model::tick() (gọi mỗi 16.67 ms – 60 FPS), rồi truyền sang View.

- Ở màn hình Fight: đọc ADC trực tiếp trong FightView::handleTickEvent() mỗi frame.

⇒ Tốc độ thực tế: ~120 mẫu/giây (2 mẫu mỗi 16.67 ms)

* LCD

- Dùng LTDC + SDRAM để hiển thị đồ họa.

- TouchGFX cập nhật giao diện mỗi frame theo VSync.

⇒ Tốc độ làm mới: 60 FPS (mỗi 16.67 ms)

* FreeRTOS

- Tick scheduler cấu hình tại configTICK\_RATE\_HZ = 1000.

⇒ Tốc độ tick thực tế: 1000 tick/giây (1 ms mỗi tick)

## - Thiết kế phần mềm:

### +) Module điều khiển joystick:

* Thư viện:
* **#include** "stm32f4xx\_hal.h"
* **#include** "stm32f4xx\_hal\_adc.h"
* **ADC\_HandleTypeDef** hadc1;
* **ADC\_HandleTypeDef** hadc2;
* Model tick:
* **void** **Model::tick**()
* {
* **HAL\_ADC\_Start**(&hadc1);
* **HAL\_ADC\_PollForConversion**(&hadc1, 10);
* **int** xADC = **HAL\_ADC\_GetValue**(&hadc1);
* **HAL\_ADC\_Stop**(&hadc1);
* **HAL\_ADC\_Start**(&hadc2);
* **HAL\_ADC\_PollForConversion**(&hadc2, 10);
* **int** yADC = **HAL\_ADC\_GetValue**(&hadc2);
* **HAL\_ADC\_Stop**(&hadc2);
* **if** (modelListener)
* {
* modelListener->updatePointerPosition(xADC, yADC); // ← truyền nguyên giá trị ADC
* }
* }
* Cấu hình GPIO\_Init
* \_\_HAL\_RCC\_ADC1\_CLK\_ENABLE();
* \_\_HAL\_RCC\_ADC2\_CLK\_ENABLE();

/\* Configure PC3 (ADC1 Channel 13) as analog \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_3;

GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

**HAL\_GPIO\_Init**(GPIOC, &GPIO\_InitStruct);

/\* Configure PA5 (ADC2 Channel 5) as analog \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_5;

GPIO\_InitStruct.Mode = GPIO\_MODE\_ANALOG;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

**HAL\_GPIO\_Init**(GPIOA, &GPIO\_InitStruct);

/\* Configure PA0 (USER button) as input \*/

GPIO\_InitStruct.Pin = GPIO\_PIN\_0;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_NOPULL; // hoặc GPIO\_PULLDOWN nếu cần trạng thái rõ ràng

**HAL\_GPIO\_Init**(GPIOA, &GPIO\_InitStruct);

* Màn hình StartGame:
* **void** **StartGameView::setupScreen**()
* {
* **StartGameViewBase**::**setupScreen**();
* pointerX = 120;
* pointerY = 100;
* pointerImage.setXY(pointerX, pointerY);
* pointerImage.**invalidate**();
* }
* **void** **StartGameView::updatePointerPosition**(**int** xADC, **int** yADC)
* {
* **int** deltaX = xADC - 128;
* **int** deltaY = 128 - yADC;
* pointerX += deltaX / 12;
* pointerY += deltaY / 12;
* pointerX = std::**max**(0, std::**min**(pointerX, 225));
* pointerY = std::**max**(0, std::**min**(pointerY, 300));
* pointerImage.**invalidate**();
* pointerImage.setXY(pointerX, pointerY);
* pointerImage.**invalidate**();
* //Giới hạn vùng của flexButton1 (cố định theo TouchGFX Designer)
* **int** btnX = 84; // = 0
* **int** btnY = 177; // = 160
* **int** btnW = 72; // = 240
* **int** btnH = 20; // = 50
* **bool** isPointerInside =
* pointerX >= btnX && pointerX <= (btnX + btnW) &&
* pointerY >= btnY && pointerY <= (btnY + btnH);
* //Nếu nằm trong vùng nút và người dùng ấn nút
* **if** (isPointerInside && **HAL\_GPIO\_ReadPin**(GPIOA, GPIO\_PIN\_0) == *GPIO\_PIN\_SET*)
* {
* application().**gotoMapScreenNoTransition**();
* }
* }
* Màn hình MAP (Di chuyển nhân vật)
* // Tọa độ ban đầu của npc1 (nữ)
* **int** npc1X = 54;
* **int** npc1Y = 30;
* **MapView::MapView**()
* {
* }
* **void** **MapView::setupScreen**()
* {
* **MapViewBase**::**setupScreen**();
* // Khởi tạo vị trí npc1
* npc1X = 54;
* npc1Y = 60;
* npc1.setXY(npc1X, npc1Y);
* npc1.**invalidate**();
* }
* **void** **MapView::tearDownScreen**()
* {
* **MapViewBase**::tearDownScreen();
* }
* **void** **MapView::updatePointerPosition**(**int** xADC, **int** yADC)
* {
* **int** deltaX = xADC - 128;
* **int** deltaY = 128 - yADC;
* npc1X += deltaX / 12;
* npc1Y += deltaY / 12;
* npc1X = std::**max**(0, std::**min**(npc1X, 200));
* npc1Y = std::**max**(0, std::**min**(npc1Y, 288));
* npc1.**invalidate**();
* npc1.setXY(npc1X, npc1Y);
* npc1.**invalidate**();
* // === Thông số cố định của npc2 ===
* **const** **int** npc2X = 160;
* **const** **int** npc2Y = 71;
* **const** **int** npcW = 30;
* **const** **int** npcH = 30;
* // === Kiểm tra va chạm giữa npc1 và npc2 ===
* **bool** isColliding =
* npc1X + npcW > npc2X &&
* npc1X < npc2X + npcW &&
* npc1Y + npcH > npc2Y &&
* npc1Y < npc2Y + npcH;
* **if** (isColliding)
* {
* application().**gotoFightScreenNoTransition**();
* }
* }
* Màn hình FIGHT (Di chuyển như con chuột)
* **void** **FightView::handleTickEvent**()
* {
* **FightViewBase**::handleTickEvent();
* **HAL\_ADC\_Start**(&hadc1);
* **HAL\_ADC\_PollForConversion**(&hadc1, 10);
* **uint16\_t** xADC = **HAL\_ADC\_GetValue**(&hadc1);
* **HAL\_ADC\_Stop**(&hadc1);
* **HAL\_ADC\_Start**(&hadc2);
* **HAL\_ADC\_PollForConversion**(&hadc2, 10);
* **uint16\_t** yADC = **HAL\_ADC\_GetValue**(&hadc2);
* **HAL\_ADC\_Stop**(&hadc2);
* // Tính độ lệch và cập nhật vị trí con trỏ
* **int** deltaX = xADC - 128;
* **int** deltaY = 128 - yADC;
* **int** x = mouse.getX();
* **int** y = mouse.getY();
* x += deltaX / 12;
* y += deltaY / 12;
* x = std::**max**(0, std::**min**(x, 240));
* y = std::**max**(0, std::**min**(y, 320));
* mouse.**invalidate**();
* mouse.setXY(x, y);
* mouse.**invalidate**();
* // Nếu người dùng nhấn nút (VD: nút USER trên board)
* **if** (**HAL\_GPIO\_ReadPin**(GPIOA, GPIO\_PIN\_0) == *GPIO\_PIN\_SET*) {
* **onCursorClicked**(); // Gọi luôn xử lý click
* }
* **void** **FightView::onCursorClicked**()
* {
* // Lấy tọa độ con trỏ
* **int** cx = mouse.getX();
* **int** cy = mouse.getY();
* // Kiểm tra va chạm với button FIGHT
* **if** (FightButton.isVisible() && FightButton.**getAbsoluteRect**().intersect(cx, cy))
* {
* // Gọi hàm xử lý fight
* **Fight**();
* **return**;
* }
* // Kiểm tra va chạm với button BAG
* **if** (BagButton.isVisible() && BagButton.**getAbsoluteRect**().intersect(cx, cy))
* {
* **ShowBag**();
* **return**;
* }
* // Kiểm tra va chạm với button RUN
* **if** (runButton.isVisible() && runButton.**getAbsoluteRect**().intersect(cx, cy))
* {
* application().**gotoStartGameScreenNoTransition**();
* **return**;
* }
* **if** (level1.isVisible() && level1.**getAbsoluteRect**().intersect(cx, cy))
* {
* **Level1**();
* **return**;
* }
* **if** (level2.isVisible() && level2.**getAbsoluteRect**().intersect(cx, cy))
* {
* **Level2**();
* **return**;
* }
* **if** (level3.isVisible() && level3.**getAbsoluteRect**().intersect(cx, cy))
* {
* **Level3**();
* **return**;
* }
* **if** (Skill1Button.isVisible() && Skill1Button.**getAbsoluteRect**().intersect(cx, cy))
* {
* **FireBlast**();
* **return**;
* }
* **if** (Skill2Button.isVisible() && Skill2Button.**getAbsoluteRect**().intersect(cx, cy))
* {
* **DragonRage**();
* **return**;
* }
* **if** (Skill3Button.isVisible() && Skill3Button.**getAbsoluteRect**().intersect(cx, cy))
* {
* **Protect**();
* **return**;
* }
* **if** (FullHealButton.isVisible() && FullHealButton.**getAbsoluteRect**().intersect(cx, cy))
* {
* **useFullHeal**();
* **return**;
* }
* **if** (BomButton.isVisible() && BomButton.**getAbsoluteRect**().intersect(cx, cy))
* {
* **useBom**();
* **return**;
* }
* **if** (HideBagButton.isVisible() && HideBagButton.**getAbsoluteRect**().intersect(cx, cy))
* {
* **HideBag**();
* **return**;
* }
* **if** (PotionButton.isVisible() && PotionButton.**getAbsoluteRect**().intersect(cx, cy))
* {
* **usePotion**();
* **return**;
* }
* **if** (home.isVisible() && home.**getAbsoluteRect**().intersect(cx, cy))
* {
* application().**gotoStartGameScreenNoTransition**();
* **return**;
* }
* // Nếu không trúng gì
* }

### +) Module xử lý logic game:

* Thêm thư viện

**#include "math.h"** : Hàm cos(), sin() dùng tính toán tọa độ cho hiệu ứng bay vòng cung (bom).

**#include <cstdio>**: Hàm std::snprintf() mở rộng thành Unicode::snprintf()) để định dạng chuỗi ký tự.

* Khởi tạo màn hình (**setupScreen**):
* **void** **FightView::setupScreen**()
* {
* **FightViewBase**::**setupScreen**();
* //Khởi tạo HP
* playerHP = 150;
* BossHP = 150;
* //Khởi tạo trạng thái
* PlayerState = **false**;
* BossState = **false**;
* PlayerIsProtected = **false**;
* turn = 0;
* tickcount =0;
* **updateState**();
* PotionNum = 1;
* FullHealNum = 1;
* level = 1;
* BomNum = 1;
* **Unicode**::**snprintf**(value, 10, "%d", 150);
* HP.setWildcard(value);
* HP.**invalidate**();
* }
* Xử lý Tick Event:
* **if** (setHP){
* **Unicode**::**snprintf**(value, 10, "%d", playerHP);
* invalidate();
* setHP = **false**;
* }
* **if** (useFullHealFlashing)
* {
* useFullHealFlashCounter++;
* **if** (useFullHealFlashCounter >= 12)
* {
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **HideBag**();
* **hideActionButton**();
* PlayerState = **false**;
* useFullHealText.setVisible(**true**);
* textArea1.setVisible(**false**);
* textArea1.**invalidate**();
* useFullHealText.**invalidate**();
* turn++;
* **FullHealEffect**();
* useFullHealFlashing = **false**;
* }
* }
* **if** (usePotionFlashing)
* {
* usePotionFlashCounter++;
* **if** (usePotionFlashCounter >= 12)
* {
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **HideBag**();
* **hideActionButton**();
* playerHP = playerHP + 50;
* **if**(playerHP > 150){playerHP = 150;}
* usePotionText.setVisible(**true**);
* textArea1.setVisible(**false**);
* textArea1.**invalidate**();
* usePotionText.**invalidate**();
* turn++;
* **PlayerHealEffect**();
* usePotionFlashing = **false**;
* }
* }
* **if** (useBomFlashing)
* {
* useBomFlashCounter++;
* **if** (useBomFlashCounter >= 12)
* {
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **HideBag**();
* **hideActionButton**();
* BossHP = BossHP - 40;
* BossState = **true**;
* useBomText.setVisible(**true**);
* textArea1.setVisible(**false**);
* textArea1.**invalidate**();
* useBomText.**invalidate**();
* turn++;
* **BomEffect**();
* useBomFlashing = **false**;
* }
* }
* **if** (showFlashing)
* {
* showFlashCounter++;
* **if** (showFlashCounter >= 5)
* {
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **hideActionButton**();
* **if**(PotionNum > 0){
* PotionText.setVisible(**true**);
* PotionButton.setVisible(**true**);
* Potion.setVisible(**true**);
* PotionText.**invalidate**();
* PotionButton.**invalidate**();
* Potion.**invalidate**();
* }
* **if**(FullHealNum > 0){
* FullHealText.setVisible(**true**);
* FullHealButton.setVisible(**true**);
* FullHeal.setVisible(**true**);
* FullHealText.**invalidate**();
* FullHealButton.**invalidate**();
* FullHeal.**invalidate**();
* }
* **if**(BomNum > 0){
* Bom.setVisible(**true**);
* BomText.setVisible(**true**);
* BomButton.setVisible(**true**);
* BomText.**invalidate**();
* BomButton.**invalidate**();
* Bom.**invalidate**();
* }
* Bag.setVisible(**true**);
* Bag.**invalidate**();
* HideBagButton.setVisible(**true**);
* HideBagButton.**invalidate**();
* HideBagText.setVisible(**true**);
* HideBagText.**invalidate**();
* showFlashing = **false**;
* }
* }
* **if** (hideFlashing)
* {
* hideFlashCounter++;
* **if** (hideFlashCounter >= 5)
* {
* BomText.setVisible(**false**);
* BomButton.setVisible(**false**);
* Bom.setVisible(**false**);
* FullHealText.setVisible(**false**);
* FullHealButton.setVisible(**false**);
* PotionText.setVisible(**false**);
* PotionButton.setVisible(**false**);
* Potion.setVisible(**false**);
* FullHeal.setVisible(**false**);
* Bag.setVisible(**false**);
* HideBagText.setVisible(**false**);
* HideBagButton.setVisible(**false**);
* FightButton.setVisible(**true**);
* runButton.setVisible(**true**);
* BagButton.setVisible(**true**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* BomText.**invalidate**();
* BomButton.**invalidate**();
* Bom.**invalidate**();
* FullHealText.**invalidate**();
* FullHealButton.**invalidate**();
* PotionText.**invalidate**();
* PotionButton.**invalidate**();
* Potion.**invalidate**();
* FullHeal.**invalidate**();
* Bag.**invalidate**();
* HideBagText.**invalidate**();
* HideBagButton.**invalidate**();
* hideFlashing = **false**;
* }
* }
* **if** (bossFlashing)
* {
* bossFlashCounter++;
* **if** (bossFlashCounter % 5 == 0 && bossFlashState < 6) // Nhấp nháy 3 lần
* {
* **bool** visible = pokemon2.isVisible();
* pokemon2.setVisible(!visible);
* pokemon2.**invalidate**();
* bossFlashState++;
* }
* **if** (bossFlashState >= 6)
* {
* bossFlashing = **false**;
* **updateHP**();
* **updateState**();
* **CheckEnd**();
* **Delay**();
* **SwapPointer**();
* }
* }
* **if** (playerFlashing)
* {
* playerFlashCounter++;
* **if** (playerFlashCounter % 5 == 0 && playerFlashState < 6) // Nhấp nháy 3 lần
* {
* **bool** visible\_player = pokemon1.isVisible();
* pokemon1.setVisible(!visible\_player);
* pokemon1.**invalidate**();
* playerFlashState++;
* }
* **if** (playerFlashState >= 6)
* {
* playerFlashing = **false**;
* **if**(BossHP > 0){
* **CheckEnd**();
* **updateHP**();
* **updateState**();
* **SwapPointer**();
* }
* FightButton.setVisible(**true**);
* runButton.setVisible(**true**);
* BagButton.setVisible(**true**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* }
* }
* **if** (delay)
* {
* delayCounter++;
* **if** (delayCounter >= 15)
* {
* delay = **false**;
* **BossAttack**();
* }
* }
* **if** (fireFlashing)
* {
* fireFlashCounter++;
* **if** (fireFlashCounter % 10 == 0 && fireFlashState < 4) // Nhấp nháy 2 lần
* {
* **bool** visible1 = fire1.isVisible();
* **bool** visible2 = fire2.isVisible();
* **bool** visible3 = fire3.isVisible();
* fire1.setVisible(!visible1);
* fire2.setVisible(!visible2);
* fire3.setVisible(!visible3);
* fire1.**invalidate**();
* fire2.**invalidate**();
* fire3.**invalidate**();
* fireFlashState++;
* }
* **if** (fireFlashCounter >= 60) // Sau khi nhấp nháy xong
* {
* fireFlashing = **false**;
* textArea1.setVisible(**true**);
* Skill1Text.setVisible(**false**);
* Skill1Text.**invalidate**();
* textArea1.**invalidate**();
* **BossToggle**();
* }
* }
* **if** (shieldFlashing)
* {
* shieldFlashCounter++;
* **if** (shieldFlashCounter % 15 == 0 && shieldFlashState < 4)
* {
* **bool** visible4 = shiled.isVisible();
* shiled.setVisible(!visible4);
* shiled.**invalidate**();
* shieldFlashState++;
* }
* **if** (shieldFlashCounter >= 60)
* {
* shieldFlashing = **false**;
* textArea1.setVisible(**true**);
* Skill3Text.setVisible(**false**);
* Skill3Text.**invalidate**();
* textArea1.**invalidate**();
* **Delay**();
* **SwapPointer**();
* }
* }
* **if** (shieldFlashing2)
* {
* shieldFlashCounter2++;
* **if** (shieldFlashCounter2 % 15 == 0 && shieldFlashState2 < 4)
* {
* **bool** visible\_shield = shieldBreak.isVisible();
* shieldBreak.setVisible(!visible\_shield);
* shieldBreak.**invalidate**();
* shieldFlashState2++;
* }
* **if** (shieldFlashCounter2 >= 60)
* {
* shieldFlashing2 = **false**;
* ProtectText.setVisible(**false**);
* textArea1.setVisible(**true**);
* textArea1.**invalidate**();
* ProtectText.**invalidate**();
* **SwapPointer**();
* FightButton.setVisible(**true**);
* runButton.setVisible(**true**);
* BagButton.setVisible(**true**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* }
* }
* **if** (stunFlashing)
* {
* stunFlashCounter++;
* **if** (stunFlashCounter % 15 == 0 && stunFlashState < 6) // Nhấp nháy 3 lần
* {
* **bool** visible5 = stun.isVisible();
* stun.setVisible(!visible5);
* stun.**invalidate**();
* stunFlashState++;
* }
* **if** (stunFlashCounter >= 120)
* {
* stunFlashing = **false**;
* textArea1.setVisible(**true**);
* StunText.setVisible(**false**);
* StunText.**invalidate**();
* textArea1.**invalidate**();
* **SwapPointer**();
* FightButton.setVisible(**true**);
* runButton.setVisible(**true**);
* BagButton.setVisible(**true**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* }
* }
* **if** (fearFlashing)
* {
* fearFlashCounter++;
* **if** (fearFlashCounter % 15 == 0 && fearFlashState < 6) // Nhấp nháy 3 lần
* {
* **bool** visible6 = fear.isVisible();
* fear.setVisible(!visible6);
* fear.**invalidate**();
* fearFlashState++;
* }
* **if** (fearFlashCounter >= 120)
* {
* fearFlashing = **false**;
* textArea1.setVisible(**true**);
* FearText.setVisible(**false**);
* FearText.**invalidate**();
* textArea1.**invalidate**();
* **Delay**();
* **SwapPointer**();
* }
* }
* **if** (bomFlashing)
* {
* bomFlashCounter++;
* **if** (bomFlashCounter % 1 == 0 && bomFlashState == 0 )
* {
* // Cập nhật góc
* arcAngle = arcAngle+arcStep;
* **if** (arcAngle >= 3.1415f\*3/2)
* {
* // Sau khi di chuyển xong, set về vị trí ban đầu
* bom.moveTo(startX, startY);
* bom.**invalidate**();
* bomFlashState++;
* }
* // Tính toạ độ mới
* **int** newX = centerX + radius \* **cos**(arcAngle);
* **int** newY = centerY + radius \* **sin**(arcAngle);
* bom.moveTo(newX, newY);
* bom.setVisible(**true**);
* bom.**invalidate**();
* }
* **if** ( bomFlashCounter % 45 == 0 && bomFlashState == 1 )
* {
* phatNo.setVisible(**true**);
* phatNo.**invalidate**();
* bomFlashState++;
* }
* **if**(bomFlashCounter % 35 ==0 && bomFlashState == 2){
* bomFlashing = **false**;
* bom.setVisible(**false**);
* bom.**invalidate**();
* phatNo.setVisible(**false**);
* phatNo.**invalidate**();
* useBomText.setVisible(**false**);
* textArea1.setVisible(**true**);
* textArea1.**invalidate**();
* useBomText.**invalidate**();
* **BossToggle**();
* }
* }
* **if** (thunderFlashing)
* {
* thunderFlashCounter++;
* cloud.setVisible(**true**);
* cloud.**invalidate**();
* **if** (thunderFlashCounter % 5 == 0 && thunderFlashState < 6)
* {
* **bool** visibleThunder = thunder.isVisible();
* thunder.setVisible(!visibleThunder);
* thunder.**invalidate**();
* thunderFlashState++;
* }
* **if** (thunderFlashCounter >= 60)
* {
* thunderFlashing = **false**;
* cloud.setVisible(**false**);
* cloud.**invalidate**();
* textArea1.setVisible(**true**);
* ThunderText.setVisible(**false**);
* ThunderText.**invalidate**();
* textArea1.**invalidate**();
* **PlayerToggle**();
* }
* }
* **if** (beamFlashing)
* {
* beamFlashCounter++;
* DragonBeam.setVisible(**true**);
* DragonBeam.**invalidate**();
* **if** (beamFlashCounter % 15 == 0 )
* {
* vaCham.setVisible(**true**);
* vaCham.**invalidate**();
* }
* **if** (beamFlashCounter >= 60)
* {
* beamFlashing = **false**;
* DragonBeam.setVisible(**false**);
* DragonBeam.**invalidate**();
* vaCham.setVisible(**false**);
* vaCham.**invalidate**();
* textArea1.setVisible(**true**);
* Skill2Text.setVisible(**false**);
* Skill2Text.**invalidate**();
* textArea1.**invalidate**();
* **BossToggle**();
* }
* }
* **if** (eyeFlashing)
* {
* eyeFlashCounter++;
* DragonEye2.setVisible(**true**);
* DragonEye2.**invalidate**();
* **if** (eyeFlashCounter == 30 )
* {
* DragonEye.setVisible(**true**);
* DragonEye.**invalidate**();
* }
* **if** (eyeFlashCounter >= 60)
* {
* eyeFlashing = **false**;
* DragonEye.setVisible(**false**);
* DragonEye.**invalidate**();
* DragonEye2.setVisible(**false**);
* DragonEye2.**invalidate**();
* textArea1.setVisible(**true**);
* DragonFearText.setVisible(**false**);
* DragonFearText.**invalidate**();
* textArea1.**invalidate**();
* **PlayerToggle**();
* }
* }
* **if** (healFlashing)
* {
* healFlashCounter++;
* **if** (healFlashCounter % 10 == 0 && healFlashState < 6)
* {
* **bool** visible\_heal = heal.isVisible();
* heal.setVisible(!visible\_heal);
* heal.**invalidate**();
* healFlashState++;
* }
* **if** (healFlashCounter >= 60)
* {
* healFlashing = **false**;
* HealText.setVisible(**false**);
* textArea1.setVisible(**true**);
* textArea1.**invalidate**();
* HealText.**invalidate**();
* **updateHP**();
* **SwapPointer**();
* FightButton.setVisible(**true**);
* runButton.setVisible(**true**);
* BagButton.setVisible(**true**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* }
* }
* **if** (playerhealFlashing)
* {
* playerhealFlashCounter++;
* **if** (playerhealFlashCounter % 10 == 0 && playerhealFlashState < 6)
* {
* **bool** visible\_playerheal = playerHeal.isVisible();
* playerHeal.setVisible(!visible\_playerheal);
* playerHeal.**invalidate**();
* playerhealFlashState++;
* }
* **if** (playerhealFlashCounter >= 60)
* {
* playerhealFlashing = **false**;
* usePotionText.setVisible(**false**);
* textArea1.setVisible(**true**);
* textArea1.**invalidate**();
* usePotionText.**invalidate**();
* **updateHP**();
* **Delay**();
* **SwapPointer**();
* }
* }
* **if** (fullhealFlashing)
* {
* fullhealFlashCounter++;
* **if** (fullhealFlashCounter % 10 == 0 && fullhealFlashState < 6)
* {
* **bool** visible\_fullheal = purification.isVisible();
* purification.setVisible(!visible\_fullheal);
* purification.**invalidate**();
* fullhealFlashState++;
* }
* **if** (fullhealFlashCounter >= 60)
* {
* fullhealFlashing = **false**;
* useFullHealText.setVisible(**false**);
* textArea1.setVisible(**true**);
* textArea1.**invalidate**();
* useFullHealText.**invalidate**();
* **updateState**();
* **Delay**();
* **SwapPointer**();
* }
* }
* }

 **Điều khiển hiệu ứng nhấp nháy (flashing)** cho từng hành động hay kỹ năng.

 **Điều khiển hiển thị/ẩn các nút** (Fight, Run, Bag, các icon item…) và các thông báo text (ví dụ “You used Potion”).

 **Cập nhật giá trị HP**, turn, và các hàm hiệu ứng (như FullHealEffect(), PlayerHealEffect(), BomEffect()…) khi đến mốc thời gian phù hợp.

 **Chuyển trạng thái game** (ví dụ chuyển sang màn đánh, gọi gotoFightScreenNoTransition() khi NPC va chạm, hay gọi BossAttack() sau một khoảng delay).

 **Luồng hoạt động chung của nhấp nháy**

**Counter tăng dần** (ví dụ usePotionFlashCounter++).

**Kiểm tra đạt ngưỡng** (thường >= N, hoặc counter % M == 0 && state < K) để:

* Thực hiện **nhấp nháy** (toggle visibility + invalidate) một số thành phần, đếm số lần nhấp nháy giới hạn (ví dụ 3 lần nhấp → state < 6 với % 5).
* Khi tới mốc cuối (counter đủ lớn hoặc đã nhấp nháy xong), **thực thi logic chính** của hành động:
  + Cập nhật HP (tăng/bớt và clamp cho hợp lệ).
  + Chuyển đổi turn (turn++).
  + Hiện văn bản “You used X” (các useXText.setVisible(true)), ẩn textArea1.
  + Gọi hàm hiệu ứng (PlayerHealEffect(), BomEffect(), …).
  + Reset cờ useXFlashing = false để không lặp lại.
* **setHP**
  + Ngay khi có thay đổi playerHP, in ra con số mới vào value[] và gọi invalidate() để vẽ lại HP bar/text.
* **Dùng item**
  + useFullHealFlashing (full heal): sau 12 tick, tăng turn, gọi FullHealEffect().
  + usePotionFlashing (potion): tương tự, cộng 50 HP (clamp ≤150), gọi PlayerHealEffect().
  + useBomFlashing (bomb): trừ 40 HP đối thủ, đặt BossState=true, gọi BomEffect().
* **Mở/đóng túi đồ (Bag UI)**
  + showFlashing: sau 5 tick, hiện các nút/item trong túi (Potion, FullHeal, Bom…) và nút HideBag.
  + hideFlashing: sau 5 tick, ẩn hết UI túi và hiện lại 3 nút chính (Fight, Run, Bag).
* **Hiệu ứng va chạm & đòn đánh**
  + bossFlashing: boss nhấp nháy 3 lần để báo hiệu tấn công → sau đó gọi updateHP(), updateState(), CheckEnd(), Delay(), SwapPointer().
  + playerFlashing: player nhấp nháy khi bị đánh → sau đó nếu boss còn sống thì CheckEnd()… rồi hiện lại nút hành động.
* **Delay trước đòn tấn công của boss**
  + delay: tăng delayCounter; khi đạt 15 → tắt delay và gọi BossAttack().
* **Các kỹ năng đặc biệt**
  + fireFlashing: nhấp nháy lửa 2 lần (fire1/2/3) rồi call BossToggle().
  + shieldFlashing & shieldFlashing2: nhấp nháy khi dùng shield và shield break, rồi reveal text + SwapPointer().
  + stunFlashing, fearFlashing: nhấp nháy icon stun/fear 3 lần → text + SwapPointer()/Delay().
  + bomFlashing:
    - Vẽ quỹ đạo cung tròn cho bom (cos/sin, arcAngle tăng) rồi explosion.
    - Hiện phatNo (hình nổ).
    - Sau hoàn thành, ẩn bomber, gọi BossToggle().
  + thunderFlashing: hiển thị mây, nhấp nháy icon thunder 3 lần, rồi ẩn, gọi PlayerToggle().
  + beamFlashing: hiện beam và va chạm, sau 60 tick → ẩn, gọi BossToggle().
  + eyeFlashing: hiện DragonEye2, sau 30 tick hiện DragonEye, sau 60 tick → ẩn, gọi PlayerToggle().
  + healFlashing: nhấp nháy icon heal 3 lần, sau 60 tick → ẩn text, updateHP(), SwapPointer().
  + playerhealFlashing: tương tự cho heal của player.
  + fullhealFlashing: tương tự cho full heal, sau đó updateState(), Delay(), SwapPointer().
* Ẩn các nút Action
* **void** **FightView::hideActionButton**()
* {
* Skill\_3.setVisible(**false**);
* Skill3Button.setVisible(**false**);
* Skill\_2.setVisible(**false**);
* Skill2Button.setVisible(**false**);
* Skill\_1.setVisible(**false**);
* Skill1Button.setVisible(**false**);
* boxWithBorder2.setVisible(**false**);
* Skill\_3.**invalidate**();
* Skill3Button.**invalidate**();
* Skill\_2.**invalidate**();
* Skill2Button.**invalidate**();
* Skill\_1.**invalidate**();
* Skill1Button.**invalidate**();
* boxWithBorder2.**invalidate**();
* }

=> Ẩn hoàn toàn ba nút skill và khung chứa, đảm bảo người chơi không thể chọn skill khi chưa vào lượt.

* Chọn cấp độ chơi
* **void** **FightView::Level1**(){
* level = 1;
* selectLevel.setVisible(**false**);
* level1.setVisible(**false**);
* level2.setVisible(**false**);
* level3.setVisible(**false**);
* selectLevel.**invalidate**();
* level1.**invalidate**();
* level2.**invalidate**();
* level3.**invalidate**();
* }
* **void** **FightView::Level2**(){
* level = 2;
* selectLevel.setVisible(**false**);
* level1.setVisible(**false**);
* level2.setVisible(**false**);
* level3.setVisible(**false**);
* selectLevel.**invalidate**();
* level1.**invalidate**();
* level2.**invalidate**();
* level3.**invalidate**();
* }
* **void** **FightView::Level3**(){
* level = 3;
* selectLevel.setVisible(**false**);
* level1.setVisible(**false**);
* level2.setVisible(**false**);
* level3.setVisible(**false**);
* selectLevel.**invalidate**();
* level1.**invalidate**();
* level2.**invalidate**();
* level3.**invalidate**();
* }
* Hiện bảng chọn Skill
* **void** **FightView::Fight**()
* {
* Skill\_3.setVisible(**true**);
* Skill3Button.setVisible(**true**);
* Skill\_2.setVisible(**true**);
* Skill2Button.setVisible(**true**);
* Skill\_1.setVisible(**true**);
* Skill1Button.setVisible(**true**);
* boxWithBorder2.setVisible(**true**);
* Skill\_3.**invalidate**();
* Skill3Button.**invalidate**();
* Skill\_2.**invalidate**();
* Skill2Button.**invalidate**();
* Skill\_1.**invalidate**();
* Skill1Button.**invalidate**();
* boxWithBorder2.**invalidate**();
* }
* Các chiêu thức người chơi
  + Fire Blast:
* **void** **FightView::FireBlast**()
* {
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **if**(PlayerState == **true**){
* **int** x = **Random**(1,10);
* **if**(x<=4){
* **hideActionButton**();
* FearText.setVisible(**true**);
* textArea1.setVisible(**false**);
* FearText.**invalidate**();
* textArea1.**invalidate**();
* **FearToggle**();
* PlayerState = **false**;
* **updateState**();
* turn++;
* **return**;
* }**else**{
* **hideActionButton**();
* FearText.setVisible(**true**);
* textArea1.setVisible(**false**);
* FearText.**invalidate**();
* textArea1.**invalidate**();
* **FearToggle**();
* turn++;
* **return**;
* }
* }**else**
* **hideActionButton**();
* textArea1.setVisible(**false**);
* Skill1Text.setVisible(**true**);
* Skill1Text.**invalidate**();
* textArea1.**invalidate**();
* **ShowFire**();
* BossHP = BossHP - 30;
* //BossToggle();
* turn++;
* }
  + Dragon Rage:
* **void** **FightView::DragonRage**()
* {
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **if**(PlayerState == **true**){
* **hideActionButton**();
* FearText.setVisible(**true**);
* textArea1.setVisible(**false**);
* FearText.**invalidate**();
* textArea1.**invalidate**();
* **FearToggle**();
* PlayerState = **false**;
* **updateState**();
* turn++;
* **return**;
* }**else**
* **hideActionButton**();
* textArea1.setVisible(**false**);
* Skill2Text.setVisible(**true**);
* Skill2Text.**invalidate**();
* textArea1.**invalidate**();
* BossHP = BossHP - 20;
* **DragonRageEffect**();
* **int** x = **Random**(1,10);
* **if**(x<=5){
* BossState = **true**;
* }
* turn++;
* }
  + Protect:
* **void** **FightView::Protect**()
* {
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **if**(PlayerState == **true**){
* **hideActionButton**();
* FearText.setVisible(**true**);
* textArea1.setVisible(**false**);
* FearText.**invalidate**();
* textArea1.**invalidate**();
* **FearToggle**();
* PlayerState = **false**;
* **updateState**();
* turn++;
* **return**;
* }**else**
* **hideActionButton**();
* textArea1.setVisible(**false**);
* Skill3Text.setVisible(**true**);
* Skill3Text.**invalidate**();
* textArea1.**invalidate**();
* **shieldToggle**();
* PlayerIsProtected = **true**;
* shiled.setVisible(**true**);
* shiled.**invalidate**();
* turn++;
* }
* Vật phẩm items
* **void** **FightView::useBom**(){
* BomNum = BomNum -1;
* **if**(BomNum == 0){
* **useBomDelay**();
* }
* }
* **void** **FightView::usePotion**(){
* PotionNum = PotionNum - 1;
* **if**(PotionNum == 0){
* **usePotionDelay**();
* }
* }
* **void** **FightView::useFullHeal**(){
* FullHealNum = FullHealNum - 1;
* **if**(FullHealNum == 0){
* **useFullHealDelay**();
* }
* }
* Các chiêu thức của Boss
* **void** **FightView::Thunder**()
* {
* **if**(PlayerIsProtected == **true**){
* ProtectText.setVisible(**true**);
* textArea1.setVisible(**false**);
* ProtectText.**invalidate**();
* textArea1.**invalidate**();
* **shieldToggle2**();
* PlayerIsProtected = **false**;
* **updateState**();
* **return**;
* }**else**
* playerHP = playerHP - 30;
* textArea1.setVisible(**false**);
* ThunderText.setVisible(**true**);
* ThunderText.**invalidate**();
* textArea1.**invalidate**();
* **ThunderEffect**();
* }
* **void** **FightView::DragonFear**(){
* **if**(PlayerIsProtected == **true**){
* ProtectText.setVisible(**true**);
* textArea1.setVisible(**false**);
* ProtectText.**invalidate**();
* textArea1.**invalidate**();
* **shieldToggle2**();
* PlayerIsProtected = **false**;
* **updateState**();
* **return**;
* }**else**
* playerHP = playerHP - 10;
* PlayerState = **true**;
* textArea1.setVisible(**false**);
* DragonFearText.setVisible(**true**);
* DragonFearText.**invalidate**();
* textArea1.**invalidate**();
* **DragonFearEffect**();
* }
* **void** **FightView::Heal**(){
* BossHP = BossHP + 50;
* **if**(BossHP > 150){BossHP = 150;}
* textArea1.setVisible(**false**);
* HealText.setVisible(**true**);
* HealText.**invalidate**();
* textArea1.**invalidate**();
* **HealEffect**();
* }
* Random: Đây là một hàm sinh số ngẫu nhiên dựa trên thuật toán **Linear Congruential Generator (LCG)**
* **int** **FightView::Random**(**int** min,**int** max){
* randomseed = (randomseed \* 1103515245 + 12345) & 0x7fffffff;
* **return** min + (randomseed % (max - min + 1));
* }

**1. Cập nhật randomseed (LCG step)**

* **Công thức**:

randomseed←(a×randomseed+c) mod m

với

* + a =1103515245 (multiplier)
  + c =12345 (increment)
  + m =
* &0x7fffffff để đảm bảo rằng kết quả luôn là số dương

**2. Phổ biến về [min, max]**

Sau khi cập nhật seed, hàm trả ra:

min + (randomseed % (max - min + 1))

* randomseed % (max - min + 1) đưa kết quả về một số trong khoảng [0…(max−min)]
* Cộng thêm min chuyển thành khoảng [min…max]
* Cập nhật HP, State, Checkend
* **void** **FightView::updateHP**()
* {
* boxProgress1.**setValue**(BossHP);
* boxProgress1\_2.**setValue**(playerHP);
* setHP = **true**;
* }
* **void** **FightView::updateState**()
* {
* stun.setVisible(BossState);
* fear.setVisible(PlayerState);
* shiled.setVisible(PlayerIsProtected);
* stun.**invalidate**();
* fear.**invalidate**();
* shiled.**invalidate**();
* }
* **void** **FightView::CheckEnd**(){
* **if**(playerHP <= 0){
* Defeat.setVisible(**true**);
* home.setVisible(**true**);
* Defeat.**invalidate**();
* home.**invalidate**();
* **return**;
* }**else** **if**(BossHP <= 0){
* Victory.setVisible(**true**);
* home.setVisible(**true**);
* Victory.**invalidate**();
* home.**invalidate**();
* **return**;
* }
* }
* Các phương thức khác
  + Swap Pointer
* **void** **FightView::SwapPointer**(){
* PlayerPointer.setVisible(BossPointer.isVisible());
* BossPointer.setVisible(!BossPointer.isVisible());
* PlayerPointer.**invalidate**();
* BossPointer.**invalidate**();
* }
  + Delay
* **void** **FightView::hideBagDelay**(){
* hideFlashing = **true**;
* hideFlashCounter = 0;
* }
* **void** **FightView::showBagDelay**(){
* showFlashing = **true**;
* showFlashCounter = 0;
* }
* **void** **FightView::useBomDelay**(){
* useBomFlashing = **true**;
* useBomFlashCounter = 0;
* }
* **void** **FightView::usePotionDelay**(){
* usePotionFlashing = **true**;
* usePotionFlashCounter = 0;
* }
* **void** **FightView::useFullHealDelay**(){
* useFullHealFlashing = **true**;
* useFullHealFlashCounter = 0;
* }
  + Toggle
* **void** **FightView::BossToggle**()
* {
* bossFlashing = **true**;
* bossFlashCounter = 0;
* bossFlashState = 0;
* }
* **void** **FightView::PlayerToggle**()
* {
* playerFlashing = **true**;
* playerFlashCounter = 0;
* playerFlashState = 0;
* }
* **void** **FightView::ShowFire**(){
* fireFlashing = **true**;
* fireFlashCounter = 0;
* fireFlashState = 0;
* }
* **void** **FightView::DragonRageEffect**(){
* beamFlashing = **true**;
* beamFlashCounter = 0;
* }
* **void** **FightView::StunToggle**(){
* stunFlashing = **true**;
* stunFlashCounter = 0;
* stunFlashState = 0;
* }
* **void** **FightView::FearToggle**(){
* fearFlashing = **true**;
* fearFlashCounter = 0;
* fearFlashState = 0;
* }
  + Show bag/ Hide bag
* **void** **FightView::ShowBag**(){
* FightButton.setVisible(**false**);
* runButton.setVisible(**false**);
* BagButton.setVisible(**false**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* **hideActionButton**();
* **if**(PotionNum > 0){
* PotionText.setVisible(**true**);
* PotionButton.setVisible(**true**);
* Potion.setVisible(**true**);
* PotionText.**invalidate**();
* PotionButton.**invalidate**();
* Potion.**invalidate**();
* }
* **if**(FullHealNum != 0){
* FullHealText.setVisible(**true**);
* FullHealButton.setVisible(**true**);
* FullHeal.setVisible(**true**);
* FullHealText.**invalidate**();
* FullHealButton.**invalidate**();
* FullHeal.**invalidate**();
* }
* **if**(BomNum != 0){
* Bom.setVisible(**true**);
* BomText.setVisible(**true**);
* BomButton.setVisible(**true**);
* BomText.**invalidate**();
* BomButton.**invalidate**();
* Bom.**invalidate**();
* }
* Bag.setVisible(**true**);
* Bag.**invalidate**();
* HideBagButton.setVisible(**true**);
* HideBagButton.**invalidate**();
* HideBagText.setVisible(**true**);
* HideBagText.**invalidate**();
* }
* **void** **FightView::HideBag**(){
* BomText.setVisible(**false**);
* BomButton.setVisible(**false**);
* Bom.setVisible(**false**);
* FullHealText.setVisible(**false**);
* FullHealButton.setVisible(**false**);
* PotionText.setVisible(**false**);
* PotionButton.setVisible(**false**);
* Potion.setVisible(**false**);
* FullHeal.setVisible(**false**);
* Bag.setVisible(**false**);
* HideBagText.setVisible(**false**);
* HideBagButton.setVisible(**false**);
* FightButton.setVisible(**true**);
* runButton.setVisible(**true**);
* BagButton.setVisible(**true**);
* FightButton.**invalidate**();
* runButton.**invalidate**();
* BagButton.**invalidate**();
* BomText.**invalidate**();
* BomButton.**invalidate**();
* Bom.**invalidate**();
* FullHealText.**invalidate**();
* FullHealButton.**invalidate**();
* PotionText.**invalidate**();
* PotionButton.**invalidate**();
* Potion.**invalidate**();
* FullHeal.**invalidate**();
* Bag.**invalidate**();
* HideBagText.**invalidate**();
* HideBagButton.**invalidate**();
* }
  + Bom Effect
* **void** **FightView::BomEffect**(){
* bomFlashing = **true**;
* bomFlashCounter = 0;
* bomFlashState = 0;
* // Lưu vị trí ban đầu
* startX = bom.getX();
* startY = bom.getY();
* // Thiết lập tham số quỹ đạo
* centerX = 140;
* centerY = 190;
* radius = 120;
* arcAngle = 3.1415f; // Bắt đầu từ góc 0 rad
* arcStep = 0.05f; // Góc thay đổi mỗi tick
* }

### +) Module xử lý các cấp độ level:

* Các cấp độ:
* **void** **FightView::Level1**(){
* level = 1;
* selectLevel.setVisible(**false**);
* level1.setVisible(**false**);
* level2.setVisible(**false**);
* level3.setVisible(**false**);
* selectLevel.**invalidate**();
* level1.**invalidate**();
* level2.**invalidate**();
* level3.**invalidate**();
* }
* **void** **FightView::Level2**(){
* level = 2;
* selectLevel.setVisible(**false**);
* level1.setVisible(**false**);
* level2.setVisible(**false**);
* level3.setVisible(**false**);
* selectLevel.**invalidate**();
* level1.**invalidate**();
* level2.**invalidate**();
* level3.**invalidate**();
* }
* **void** **FightView::Level3**(){
* level = 3;
* selectLevel.setVisible(**false**);
* level1.setVisible(**false**);
* level2.setVisible(**false**);
* level3.setVisible(**false**);
* selectLevel.**invalidate**();
* level1.**invalidate**();
* level2.**invalidate**();
* level3.**invalidate**();
* }
* Logic boss các cấp độ:
* **void** **FightView::BossAttack**(){
* **if**(level != 3){
* **if**(BossState == **true** ){
* StunText.setVisible(**true**);
* textArea1.setVisible(**false**);
* StunText.**invalidate**();
* textArea1.**invalidate**();
* **StunToggle**();
* BossState = **false**;
* **updateState**();
* **return** ;
* }**else** **if**(level == 1){
* **Thunder**();
* }**else** **if**(level == 2){
* **int** x = **Random**(1,10);
* **if**(x<=3){
* **DragonFear**();
* }
* **if** (BossHP <= 40){
* **Heal**();
* }**else**{
* **Thunder**();
* }
* }
* }**else**{
* **if**(BossState == **true** && turn != 1 && !(BossHP >= 40 && BossHP <= 75) && playerHP >= 30){
* StunText.setVisible(**true**);
* textArea1.setVisible(**false**);
* StunText.**invalidate**();
* textArea1.**invalidate**();
* **StunToggle**();
* BossState = **false**;
* **updateState**();
* **return**;
* }**else** **if**(turn == 1){
* BossState = **false**;
* **updateState**();
* **DragonFear**();
* }
* **else** **if**(BossHP <= 75 && BossHP >= 40){
* **int** x = **Random**(1,10);
* **if**(x<=4){
* **if**(BossState == **true** ){
* StunText.setVisible(**true**);
* textArea1.setVisible(**false**);
* StunText.**invalidate**();
* textArea1.**invalidate**();
* **StunToggle**();
* BossState = **false**;
* **updateState**();
* **return** ;
* }**else**{
* **Heal**();
* }
* }**else** **if**(x>4 && x<=6){
* **if**(BossState == **true** ){
* StunText.setVisible(**true**);
* textArea1.setVisible(**false**);
* StunText.**invalidate**();
* textArea1.**invalidate**();
* **StunToggle**();
* BossState = **false**;
* **updateState**();
* **return** ;
* }**else**{
* **Thunder**();
* }
* }**else**{
* BossState = **false**;
* **updateState**();
* **DragonFear**();
* }
* }
* **else** **if**(playerHP <= 30 ){
* BossState = **false**;
* **updateState**();
* **Thunder**();
* }
* **else** **if**(BossHP < 40 && playerHP >= 30){
* **Heal**();
* }**else**{
* **int** y = **Random**(1,10);
* **if**(y<=7){
* **Thunder**();
* }**else**{
* **DragonFear**();
* }
* }
* }

=> Giải thích chi tiết:

void FightView::BossAttack() {

if (level != 3) {

// Level 1 và 2

if (BossState) {

// Nếu boss đang bị choáng, bỏ lượt

StunText.setVisible(true);

textArea1.setVisible(false);

StunText.invalidate(); textArea1.invalidate();

StunToggle(); // bật hiệu ứng choáng

BossState = false;

updateState(); // cập nhật icon

return;

} else if (level == 1) {

// Level 1: luôn dùng Thunder

Thunder();

} else if (level == 2) {

// Level 2: mix DragonFear, Heal, Thunder

int x = Random(1,10);

if (x <= 3) {

DragonFear();

}

if (BossHP <= 40) {

Heal();

} else {

Thunder();

}

}

} else {

// Level 3: logic nâng cao

if (BossState == true && turn != 1 && !(BossHP >= 40 && BossHP <= 75) && playerHP >= 30) {

// Bị choáng trừ khi là lượt đầu hoặc HP boss trong [40,75]

StunText.setVisible(true);

textArea1.setVisible(false);

StunText.invalidate(); textArea1.invalidate();

StunToggle();

BossState = false;

updateState();

return;

} else if (turn == 1) {

// Lập trình tấn công đầu tiên của boss nhất định sử dụng DragonFear,bỏ qua mọi hiệu ứng xấu

BossState = false; updateState(); DragonFear();

} else if (BossHP >= 40 && BossHP <= 75) {

// Khi boss HP giữa 40 và 75: có 40% dùng Heal ,20% dùng Thunder và 40% dùng DragonFear

int x = Random(1,10);

if (x <= 4) {

Heal();

} else if (x>4 && x <= 6) {

Thunder();

} else {

DragonFear();

}

} else if (playerHP <= 30) {

// Nếu người chơi từ 30 trở xuống,giải toàn bộ debuff và kết liễu người chơi bằng Thunder

BossState = false; updateState(); Thunder();

} else if (BossHP < 40 && playerHP >= 30) {

// Khi boss rất yếu nhưng người chơi còn sức: ưu tiên Heal

Heal();

} else {

// Tình huống còn lại: 70% dùng Thunder, 30% DragonFear

int y = Random(1,10);

if (y <= 7) Thunder(); else DragonFear();

}

}

}

=> Cụ thể:

**Level 1**

* **Luôn** gọi Thunder(), không có Healing hay DragonFear.
* Thunder(): thực hiện animation, gây 30 damage (nếu không được bảo hộ).

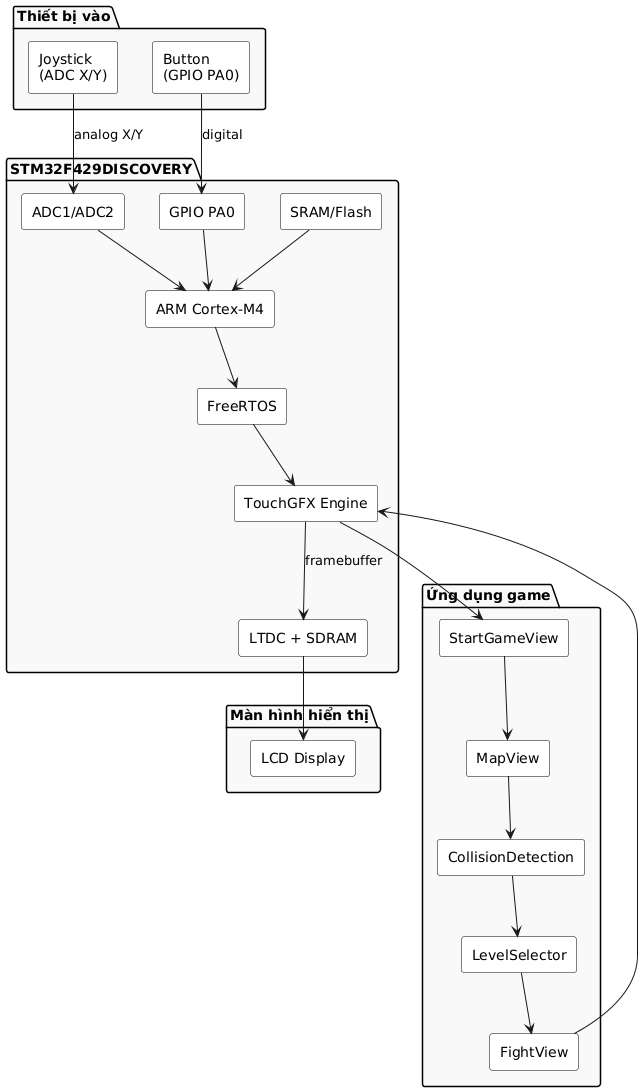
**Level 2**

* **Nếu bị choáng** (BossState): bỏ lượt.
* Ngược lại:
  + 30% cơ hội (x <= 3) dùng DragonFear() (10 damage + set PlayerState).
  + Nếu HP boss ≤ 40: gọi Heal() (+50 HP).
  + Ngược lại: dùng Thunder().

**Level 3**

* Kiểm tra BossState và turn: nếu boss bị choáng và không phải lượt 1, không nằm trong khung HP [40,75] và playerHP ≥ 30 → bỏ lượt.
* Nếu là turn == 1: khởi đầu bằng DragonFear(), bỏ qua hiệu ứng xấu,
* Nếu 40 ≤ BossHP ≤ 75:
  + 40% Heal, 20% Thunder, 40% DragonFear.
* Nếu playerHP ≤ 30: theo đuổi bằng Thunder().
* Nếu BossHP < 40 và playerHP ≥ 30: sử dụng Heal().
* Các trường hợp còn lại: 70% Thunder, 30% DragonFear.

+) Sơ đồ khối hệ thống:



**Thiết bị vào**

* **Joystick (ADC X/Y)**  
  Cung cấp hai tín hiệu analog tương ứng trục X, Y.
* **Button (GPIO PA0)**  
  Một nút nhấn đơn giản, đưa tín hiệu digital HIGH/LOW.

**STM32F429DISCOVERY (MCU)**

* **ADC1/ADC2**  
  Bộ chuyển đổi analog→digital, đọc giá trị từ joystick.
* **GPIO PA0**  
  Đọc trạng thái nút bấm.
* **ARM Cortex-M4 (CPU)**  
  Xử lý chính: nhận input, chạy game logic, gọi các service.
* **SRAM/Flash (MEM)**  
  Lưu code và giữ trạng thái game khi chạy.
* **FreeRTOS (RTOS)**  
  Quản lý nhiều task: có thể tách tác vụ đọc input, xử lý game, render riêng biệt.
* **TouchGFX Engine (TFX)**  
  Framework đồ họa: tạo widget, layout, quản lý các View/Screen.
* **LTDC + SDRAM**  
  LTDC (LCD-TFT Display Controller) kèm SDRAM để lưu framebuffer, xuất hình ra màn.

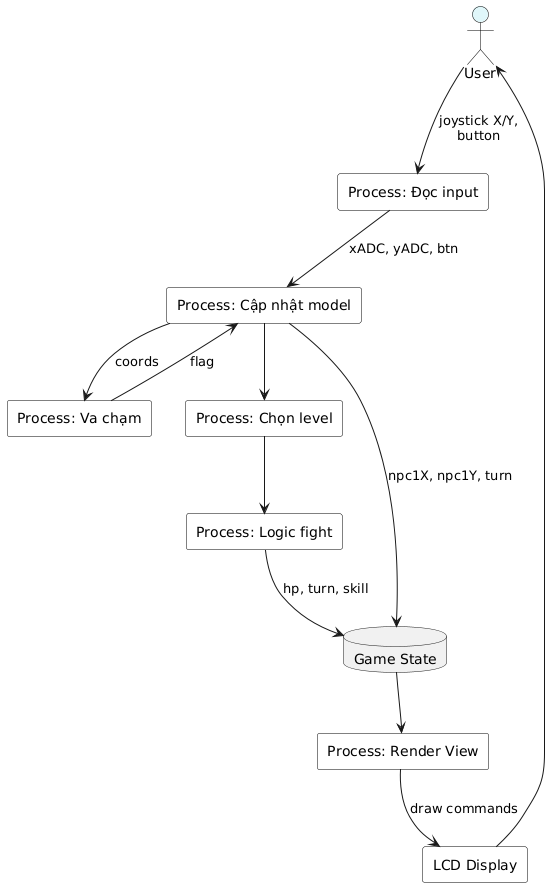
**Ứng dụng game**  
Các lớp View / Module chính do bạn tự viết, chạy trên TFX:

* **StartGameView**: màn mở đầu, nút Start.
* **MapView**: hiển thị bản đồ và con trỏ NPC.
* **CollisionDetection**: kiểm tra va chạm NPC với vật thể.
* **LevelSelector**: chọn màn (level) khi va chạm.
* **FightView**: màn đánh theo lượt.

**Màn hình hiển thị**

* **LCD Display**: hiển thị thực sự các khung hình, nội dung do LTDC xuất.

+) Sơ đồ luồng:



**User (actor)**

* Người chơi dùng joystick và nút bấm để tương tác.

**Process: Đọc input**

* Lấy giá trị xADC, yADC từ ADC và trạng thái btn từ GPIO.
* Chuyển thành dữ liệu số đưa vào quá trình kế tiếp.

**Process: Cập nhật model**

* Dựa vào xADC, yADC, btn tính toán vị trí con trỏ (npc1X, npc1Y), xử lý sự kiện nhấn nút.
* Cập nhật các biến toàn cục như turn, playerHP, v.v.
* Gửi tọa độ con trỏ (coords) tới module **Va chạm**.

**Process: Va chạm (CollisionDetection)**

* Nhận coords (tọa độ con trỏ) từ model, kiểm tra có chạm vật thể (cái tree, đá…) hay không.
* Trả về **flag** (true/false) sang **Cập nhật model** để xác định chuyển state (ví dụ ready to select level).

**Process: Chọn level (LevelSelector)**

* Khi va chạm thành công, model gọi LevelSelector để xác định level tiếp theo (theo vị trí chạm).
* Trả về thông tin level cho **Process: Logic fight**.

**Process: Logic fight (FightView)**

* Dựa vào level, khởi tạo trạng thái đánh (HP, turn, items).
* Trong mỗi lượt, update HP, hiệu ứng, gọi Random() để tính sát thương…
* Xuất dữ liệu hp, turn, skill về **Game State**.

**Game State (Data Store)**

* Nơi lưu trữ trạng thái chung của game: vị trí NPC, HP hai bên, số potion/fullheal, turn, flag thắng thua…
* Cho phép **các process** (model, fight, render) cùng đọc/ghi.

**Process: Render View**

* Đọc thông tin từ **Game State**, chuyển thành các lệnh vẽ (draw commands) như vẽ sprite, text, animation.
* Gửi lệnh này xuống LCD.

**LCD Display**

* Nhận các draw commands, vẽ lên màn hình, người chơi nhìn thấy kết quả.

# 3. Cài đặt/Xây dựng hệ thống

-Github: <https://github.com/TiepKun/PokemonFinal?fbclid=IwY2xjawLORHBleHRuA2FlbQIxMABicmlkETEyY0xiNVVwNzllbmlXQXN1AR4nyImrIoflbIX6Lrwe_zgEGHO_d9iadWz09YgeWHkBlWTcy7mid-t4ktvwKA_aem_KT-sJo3eKSWIuzvnR4fWcQ>

## - Mô tả các module phần mềm chính:

|  |  |
| --- | --- |
| Module | Chức năng chính |
| Joystick | Đọc dữ liệu joystick (ADC) & nút bấm (GPIO) |
| Map | Hiển thị bản đồ, di chuyển NPC, map scrolling |
| CollisionDetection | Kiểm tra va chạm 2NPC |
| LevelSelector | Xác định màn chơi tiếp theo khi va chạm |
| Fight | Logic đánh theo lượt, quản lý HP, hiệu ứng |

## - Đóng góp của từng thành viên

|  |  |  |  |
| --- | --- | --- | --- |
| Họ tên | MSSV | Mô tả công việc | Đánh giá |
| Nguyễn Khắc Tiếp | 20225767 | Thiết kế giao diện TouchGFX  Xử lý 3 mức độ boss. | 33% |
| Nguyễn Văn Hoàn | 20225708 | Thiết kế kỹ năng boss và người chơi,hiệu ứng kỹ năng và item. | 33% |
| Nguyễn Việt Thành | 20225931 | Xử lý ngoại vi Joystick và nút bấm. | 33% |

## - Đánh giá:

+) Đạt yêu cầu:

* Joystick hoạt động mượt mà.
* Logic game turn-based chính xác
* Boss 3 chế độ cho người chơi lựa chọn
* Thiết kế giao diện dễ nhìn

+) Ưu điểm:

* Game hoạt động ổn định
* Boss 3 cấp độ chơi đa dạng
* Kết hợp được ngoại vi Joystick để di chuyển ổn định và mượt mà hơn thay cho nút bấm vật lý

+) Nhược điểm:

* Game chưa phát triển nhiều màn chơi chỉ có một màn chơi duy nhất
* Sử dụng SDRAM lớn để lưu framebuffer, tiêu tốn nhiều RAM trên board STM32.